Clase 1 — 03.10.2025

#androidstudio

👨‍🏫 *Profesor: Joan Salvador Gordi Ortega*
📘 Programación Multimedia y Dispositivos Móviles
🗓️ Clase 1 — 03/10/2025
🎯 Tema: Presentación de la asignatura e instalación del entorno de desarrollo

---

👩‍🏫 Introducción del profesor

- El profesor explica que **intentará combinar teoría y práctica**, porque *la mejor manera de aprender a programar es programando*.
- La asignatura se desarrollará **con Android** y el lenguaje **Java**.
- Menciona que veremos **AWT o Swing**, relacionados con **eventos de formularios**.
- Pregunta si los alumnos ya conocen estos conceptos:
  → Solo hemos visto **Programación Orientada a Objetos**, **Polimorfismo**, **Herencia** y **Sobrescritura de métodos**.
  → **No hemos visto interfaces ni clases abstractas.**
- El profesor hace estas preguntas para **evaluar el punto de partida del grupo** y ajustar el ritmo del curso.
- Comenta que en la **próxima clase haremos un ejercicio de repaso** para consolidar estos fundamentos antes de avanzar hacia Android.

> 🧠 **Entender bien la POO es esencial** para desarrollar en Android, ya que gran parte del trabajo consiste en manejar eventos, sobrescribir métodos y usar clases e interfaces del sistema.

💬 Comentarios prácticos de la clase

Durante la instalación:

- El profesor muestra cómo seleccionar los componentes a instalar.
- Explica que los **emuladores pueden tardar en arrancar** y recomienda **no cerrarlos cada vez** para ahorrar tiempo.
- Se comenta la diferencia entre usar un **dispositivo físico conectado por USB** (más fluido) y un **emulador virtual** (más lento, pero más flexible para pruebas).
- También se mencionan los **requisitos mínimos del sistema** (8 GB RAM, procesador de 64 bits, 4 GB libres).

🧭 Objetivos del módulo

El profesor presenta el documento **“Desarrollo de Aplicaciones Móviles Android — Presentación del módulo”**, donde se resumen los contenidos que se verán durante el curso.

🔹 ¿Qué veremos?

- **Unidad 1 —Tecnologías móviles**
  Dispositivos, sistemas operativos y limitaciones (batería, memoria, conectividad).

- **Unidad 2 —Android Studio**
  Instalación, configuración y primeros pasos.
- **Unidad 3 — Componentes básicos**
  Introducción a *Activities*, *Intents*, *TextView*, *Button*, *ListView*.
- **Unidad 4 — Componentes de texto**
  Uso de *EditText*, *AutoCompleteTextView*, validación y captura de datos.
- **Unidad 5 — Componentes de selección**
  RadioButton, *CheckBox* y gestión de eventos `OnCheckedChangeListener`.
- **Unidad 6 — Switch**
  Control visual de dos estados (ON/OFF).
- **Unidad 7 — Imágenes**
  Uso de *ImageView*, *ImageButton* y carpetas *drawable* según densidad.
- **Unidad 8 — Listas y selección**
  Spinner, *ListView* y *ArrayAdapter*.
- **Unidad 9 — Ciclo de vida de una Activity**
  Métodos `onCreate()`, `onStart()`, `onResume()`, `onPause()`, `onStop()`, `onDestroy()`.
- **Unidad 10 — Layouts y estilos**
  LinearLayout, *TableLayout*, *FrameLayout*, *ConstraintLayout*.

📱 Unidad 1 — Tecnologías móviles

🌍 Ecosistema de dispositivos

- **Smartphones:** dispositivos multifuncionales con sensores y conectividad total.
- **Tablets:** ideales para consumo multimedia y productividad.
- **Wearables:** como smartwatches, amplían las funciones del teléfono.

⚙️ Limitaciones de los dispositivos

- **Batería limitada:** las apps deben optimizar el consumo energético.
- **Memoria restringida:** gestión eficiente de RAM.
- **Conectividad variable:** las apps deben funcionar offline y sincronizar datos al recuperar conexión.

💡 Hardware y sensores

- **Procesadores ARM:** arquitectura eficiente con bajo consumo energético.
- **Múltiples núcleos:** optimizan el rendimiento.
- **Sensores integrados:** acelerómetro, GPS y cámara son claves para muchas apps.
  - Tenemos clases y librerías que nos permiten acceder a estos elementos.

📊 Sistemas operativos móviles

- **Android (71%)** — sistema abierto basado en Linux.
- **iOS (28%)** — sistema cerrado de Apple.
- **Otros (1%)** — Windows Phone, etc., en declive.
  - Windows phone ya no es una alternativa de desarrollo. Utiliza C#, una copia de JAVA.

⚔️ Android vs iOS

Característica	Android	iOS
Código fuente	Abierto	Cerrado
Lenguaje principal	Java / Kotlin	Swift / Objective-C
Personalización	Muy alta	Limitada
Fragmentación	Alta	Baja
Tiempo de desarrollo	Más rápido	Más lento

> Cuando desarrollamos una app para android:
> Debemos elegir la version de android con la que queremos trabajar. En función de esta, nos limitara a que dispositivos con una version inferior se puede llegar a instalar.

💻 Unidad 2 — Instalación de Android Studio

📎 Enlace oficial

https://developer.android.com/studio?hl=es-419

> El profesor comenta que esta página es muy completa, con documentación sobre **Android**, **Java** y también **Kotlin**.
> Aclara que **nosotros aprenderemos con Java**, ya que es el lenguaje base y el más útil para entender los fundamentos.

⚙️ Entornos de desarrollo

El profesor explica que existen distintas plataformas para desarrollar aplicaciones móviles, pero la **mejor opción nativa** para Android es **Android Studio**, porque está creado por Google y tiene todo lo necesario integrado.

🔹 Android Studio

- Es el **IDE oficial** para Android.
- **Multiplataforma:** disponible para Windows, macOS y Linux.
- Basado en **IntelliJ IDEA**. (IntelliJ es una plataforma desarrollada por **JetBrains** (una compañía conocida por crear entornos de desarrollo de software de alta calidad para diversos lenguajes de programación.)
- Incluye:
  - **Emuladores integrados.**
  - **Depurador y herramientas de profiling.**
  - **IA integrada** que ayuda a escribir código.
- También existe **Android Studio Cloud**, una versión web en desarrollo que permitirá trabajar sin instalar nada.

> ⚠️ Android Studio es **exigente en recursos** (RAM y CPU). Los **emuladores consumen mucha memoria**, por lo que la depuración puede resultar pesada en algunos equipos.

🔹 Visual Studio Code

- Se puede usar, pero **no es recomendable** para este curso:
  - Requiere instalar múltiples extensiones.
  - Se trabaja mucho desde la consola.

- Android Studio ofrece todo integrado y accesible desde una misma interfaz.

🔹 **Eclipse**

- Este año se usará **Eclipse** como entorno principal en lugar de NetBeans, ya que es **más común en el entorno profesional**.

🧰 Instalación de Android Studio — Componentes

Durante la instalación, el profesor comenta el papel de algunos componentes:

Componente	Función
Android Emulator Hypervisor Driver	Permite hacer *pruebas en caliente*, sin apagar ni reiniciar el emulador.
Android Virtual Device (AVD)	Crea dispositivos virtuales para probar la app en distintas resoluciones, tamaños y versiones de Android.

> El profesor recalca que el ecosistema Android es **muy fragmentado**, con muchos fabricantes y versiones del sistema operativo. Por eso, los **emuladores AVD** son esenciales para comprobar la compatibilidad de nuestras apps.

🧩 Configuración inicial paso a paso

A continuación se detalla el proceso completo para realizar la **instalación y configuración inicial de Android Studio**, tal como el profesor explicó en clase y según la documentación oficial de developer.android.com/studio.

1️⃣ Descargar e instalar Android Studio

1. Acceder a la página oficial:
   👉 https://developer.android.com/studio?hl=es-419
2. Seleccionar la versión correspondiente a tu sistema operativo (**Windows, macOS o Linux**) y ejecutar el instalador.
   - El profesor menciona que **Android Studio es multiplataforma**, por lo que se puede trabajar en cualquier sistema.
   - Durante la instalación, el asistente solicitará aceptar las **licencias del SDK de Android** y confirmar los componentes a instalar.
3. Recomendaciones del profesor:
   - Instalar **en una unidad con suficiente espacio libre** (mínimo 10 GB).
   - Evitar rutas con espacios o caracteres especiales.
   - Si el equipo es limitado en RAM, **cerrar otras aplicaciones** durante la instalación y uso de emuladores.

2️⃣ Aceptar las licencias SDK

- Al ejecutar Android Studio por primera vez, se abrirá el **Asistente de configuración inicial (Setup Wizard)**.
- Este asistente guiará por los pasos de:

- **Descarga de los paquetes SDK** necesarios.
- **Aceptación de los términos y licencias de uso** de Google.
- **Selección del tipo de instalación:**
  - *Standard (recomendada)* → instala todo automáticamente.
  - *Custom (personalizada)* → permite elegir manualmente versiones y ubicaciones.

> 💡 **Consejo:** el profesor recomienda usar la instalación *Standard* en la primera configuración, para evitar conflictos de dependencias y versiones.

3️⃣ Instalar las APIs necesarias desde el SDK Manager

El **SDK Manager** permite gestionar las diferentes **versiones del sistema Android (API Levels)** y las herramientas del entorno de desarrollo.

🔹 **Acceso al SDK Manager:**

- Menú principal → **Tools > SDK Manager**
- Aquí se muestran:
  - Versiones del **Android SDK Platform** (por número de API).
  - **Herramientas del SDK** (Build-Tools, Platform-Tools, Emulator, etc.).
  - **System Images** para los emuladores.

🔹 **Qué instalar (recomendado por el profesor):**

- La **última versión estable de Android SDK Platform**.
- Las **Build-Tools** y **Platform-Tools** actualizadas.
- La **Android SDK Command-line Tools** (para tareas por terminal).
- La **System Image** de la versión de Android que se vaya a emular (por ejemplo, *Android 14 — API 34*).

🔹 **Observaciones:**

- Las versiones se identifican por su **nivel de API (API Level)**, no solo por el nombre comercial (Ejemplo: Android 14 → API 34).
- Es recomendable **mantener varias APIs instaladas** si se pretende probar la compatibilidad con dispositivos antiguos.
- Cada actualización del SDK puede requerir **nuevas licencias** que deben aceptarse antes de compilar.

4️⃣ Configurar un AVD (Android Virtual Device) desde el AVD Manager

El **AVD Manager** permite crear dispositivos virtuales (emuladores) con diferentes configuraciones de hardware y software.

🔹 **Acceso al AVD Manager:**

- Menú principal → **Tools > Device Manager** o **AVD Manager**.
- Aquí se gestionan los dispositivos virtuales existentes o se crean nuevos.

🔹 **Creación de un nuevo AVD:**

1. Hacer clic en **Create Virtual Device**.

2. Elegir el modelo de dispositivo (por ejemplo: *Pixel 7*, *Nexus 5X*, *Tablet 10"*…).
3. Seleccionar la **imagen del sistema (System Image)** según la versión Android que se desea emular (ejemplo: *Android 14 - API 34*).
4. Configurar opciones avanzadas:
   - **RAM y almacenamiento** asignados al emulador.
   - **Orientación (portrait / landscape)**.
   - **Resolución de pantalla**.
   - **Nivel de rendimiento gráfico (Hardware o Software)**.

🔹 **Componentes adicionales**

Durante la configuración, se mencionan dos herramientas importantes:

Componente	Función
Performance Android Emulator Hypervisor Driver	Permite realizar pruebas rápidas (*hot reload*) sin reiniciar el emulador. Mejora el rendimiento si el procesador soporta virtualización.
Android Virtual Device (AVD)	Contenedor virtual que simula un smartphone o tablet real con versión y características concretas.

> ⚠️ El profesor advierte que los **emuladores consumen mucha memoria RAM y CPU**, y en algunos equipos pueden ralentizar el sistema.
> En esos casos, se puede **usar un dispositivo físico** conectado por USB con el modo **Depuración USB (USB Debugging)** activado.

5️⃣ Verificar el funcionamiento correcto con un proyecto de prueba

Una vez instalado todo el entorno, se recomienda crear una aplicación básica para **verificar que Android Studio funciona correctamente**.

🔹 **Crear un nuevo proyecto**

1. En la pantalla de inicio, seleccionar **New Project**.
2. Escoger una plantilla, por ejemplo: **Empty Activity**.
3. Definir los parámetros iniciales:
   - **Name:** nombre de la app (por ejemplo, *HelloWorld*).
   - **Package name:** identificador único del proyecto (ejemplo: `com.ejemplo.helloworld`).
   - **Save location:** carpeta de destino.
   - **Language:** *Java*.
   - **Minimum SDK:** versión mínima de Android que soportará la app (por ejemplo, *API 26 – Android 8.0 Oreo*).
   - Gradle y la configuración "Build configuration language" al crear un nuevo proyecto

---

⚙️ Gradle y la configuración "Build configuration language" al crear un nuevo proyecto

Cuando en **Android Studio** seleccionamos **New Project**, uno de los pasos finales del asistente de creación muestra una opción llamada:

> **"Build configuration language"**

Esta opción se refiere al **lenguaje con el que Gradle (el sistema de construcción del proyecto)** gestionará las dependencias, compilación y empaquetado de la aplicación.

🧩 ¿Qué es Gradle?

Gradle es el **sistema de automatización de compilaciones** que usa Android Studio (y otros IDEs como IntelliJ o Eclipse) para:

- Compilar el código fuente.
- Gestionar las dependencias externas (bibliotecas).
- Generar los archivos APK o AAB (Android App Bundle).
- Firmar la aplicación antes de publicarla.
- Ejecutar tareas automatizadas (limpieza, pruebas, empaquetado, etc.).

En otras palabras, **Gradle es el motor que construye la app**.
Cada vez que presionas **"Run App" (▶️)**, Android Studio está ejecutando internamente comandos Gradle para:

1. Analizar tu código.
2. Descargar las librerías necesarias.
3. Compilar y generar el binario ejecutable.
4. Instalarlo en el emulador o dispositivo físico.

🧠 ¿Qué significa "Build configuration language"?

A partir de **Android Studio Giraffe (2023.3)**, Google introdujo una novedad importante:
ahora se puede elegir **en qué lenguaje se definen los scripts de Gradle**.

Tienes dos opciones principales:

Opción	Descripción	Extensión de archivos	Recomendación
Groovy DSL	Lenguaje clásico y predeterminado durante años en Android. Usa sintaxis más flexible, parecida a Java.	`.gradle`	Más usado en proyectos antiguos.
Kotlin DSL	Nueva opción oficial de Google. Usa el lenguaje Kotlin para escribir scripts Gradle de forma más tipada, segura y moderna.	`.gradle.kts`	Recomendado para proyectos nuevos.

🧩 Diferencia práctica

🔹 Groovy DSL (tradicional)

Los archivos se llaman `build.gradle`
La sintaxis es más suelta y menos tipada:

```
plugins {
    id 'com.android.application'
    id 'org.jetbrains.kotlin.android'
}

android {
```

```
    compileSdk 34

    defaultConfig {
        applicationId "com.example.helloworld"
        minSdk 24
        targetSdk 34
        versionCode 1
        versionName "1.0"
    }
}

dependencies {
    implementation 'androidx.core:core-ktx:1.12.0'
    implementation 'androidx.appcompat:appcompat:1.6.1'
}
```

🔹 Kotlin DSL (nuevo estándar)

Los archivos se llaman `build.gradle.kts`
Usa sintaxis de Kotlin, más clara y con autocompletado:

```
plugins {
    id("com.android.application")
    id("org.jetbrains.kotlin.android")
}

android {
    compileSdk = 34

    defaultConfig {
        applicationId = "com.example.helloworld"
        minSdk = 24
        targetSdk = 34
        versionCode = 1
        versionName = "1.0"
    }
}

dependencies {
    implementation("androidx.core:core-ktx:1.12.0")
    implementation("androidx.appcompat:appcompat:1.6.1")
}
```

> 🧠 **Diferencia clave:**
>
> - Groovy usa comillas simples y una sintaxis más parecida a XML/Java.
> - Kotlin DSL usa comillas dobles y es **más seguro** gracias al autocompletado y chequeo de tipos.

🔹 **Estructura inicial generada**

```
app/
├── src/main/java/... → Código fuente (MainActivity.java)
├── src/main/res/...  → Recursos (layouts, imágenes, strings)
```

```
├── AndroidManifest.xml  → Donde configuramos permisos
└── build.gradle → Dependencias (Librerias para conectarnos a mysql por ejemplo)
```

🔹 **Probar la app**

1. Ejecutar el proyecto con el botón ▶️ **Run App**.
2. Elegir el dispositivo de prueba:
   - **AVD configurado** → ejecuta la app en el emulador.
   - **Dispositivo físico conectado** → ejecuta directamente por USB.
3. Verificar que se muestra el texto "**Hello World!**" en pantalla.

> 💬 El profesor comenta que este paso confirma que:
>
> - El SDK y el emulador funcionan correctamente.
> - Las rutas y dependencias están configuradas.
> - Android Studio compila y ejecuta sin errores.

🧠 Consejos de optimización

- En equipos con recursos limitados:
  - Desactivar animaciones del emulador.
  - Reducir la resolución de pantalla.
  - Aumentar la memoria virtual (swap) del sistema operativo.
- Mantener Android Studio actualizado desde **Help → Check for Updates**.
- Revisar periódicamente los SDKs instalados para eliminar versiones antiguas y liberar espacio.

📂 Estructura básica de un proyecto Android

```
app/
├── src/main/java          → Código fuente Java/Kotlin
├── src/main/res           → Recursos (layouts, strings, drawables)
├── AndroidManifest.xml    → Permisos y configuración de la app
└── build.gradle           → Dependencias y configuración de compilación
```

🧱 Unidad 3 — Componentes básicos

🧩 Concepto de *Activity*

- Una **Activity** representa **una pantalla individual con interfaz de usuario**.
  - Cada pantalla es un Activity
- Es el **núcleo de la aplicación Android**, donde el usuario interactúa con la app.

Características principales

- **Pantalla única:** cada Activity controla una vista completa.
- **Interacción del usuario:** maneja clics, gestos y entrada de datos.
- **Navegación:** se comunica con otras Activities mediante **Intents**.

📨 Intents — Comunicación entre componentes

🔹 Intents Explícitos

- Especifican el componente exacto que se va a ejecutar.
- Se usan para navegar entre Activities dentro de la misma app.

```
Intent intent = new Intent(this, SecondActivity.class);
startActivity(intent);
```

🔹 Intents Implícitos

- Describen la acción sin indicar el componente exacto.
- El sistema decide qué aplicación puede manejarla.

```
Intent intent = new Intent(Intent.ACTION_VIEW, Uri.parse("https://google.com"));
startActivity(intent);
```

🎨 Componentes esenciales de interfaz

Componente	Descripción	Uso común
TextView	Muestra texto estático	Etiquetas, títulos, información
Button	Elemento interactivo que responde a clics	Acciones del usuario
ListView	Lista desplazable de elementos	Mostrar conjuntos de datos

---

🍔 Menús en Android

Tipo de menú	Descripción	Ejemplo de uso
Options Menu	Menú principal de la aplicación (ActionBar)	Ajustes generales
Context Menu	Menú contextual al mantener pulsado	Opciones de un elemento
Popup Menu	Menú flotante anclado a una vista	Acciones rápidas

---

✍️ Unidad 4 — Componentes de texto

🧾 TextView — Mostrar información

- Es el **componente básico de texto** en Android.
- Permite personalizar color, tamaño, fuente y alineación.
- Puede mostrar texto estático o dinámico.

```
<TextView
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:text="Hola Mundo"
    android:textSize="18sp"
    android:textColor="#FF0000"/>
```

🧍 EditText — Captura de datos del usuario

Características principales

1. **Tipos de entrada:** texto, número, email, contraseña, multilinea, etc.
2. **Validación:** *hints*, límites de caracteres, filtros y comprobaciones en tiempo real.
3. **Eventos:** uso de `TextWatcher` o `OnEditorAction` para detectar cambios.

💡 AutoCompleteTextView — Sugerencias inteligentes

- Extiende *EditText* para mostrar **sugerencias automáticas** mientras el usuario escribe.
- Mejora la experiencia de entrada y reduce errores.

Elementos clave

- **ArrayAdapter** personalizado con los datos sugeridos.
- **Filtrado automático** según los caracteres escritos.
- **Selección táctil:** el clic en una sugerencia completa el campo.

📋 Captura de datos desde la interfaz

Pasos comunes en una app Android:

1. **Obtener referencia al componente:**

```
EditText nombre = findViewById(R.id.txtNombre);
```

2. **Extraer el texto como String:**

```
String valor = nombre.getText().toString();
```

3. **Validar los datos:**
   Comprobar formato, longitud y contenido.
4. **Procesar la información:**
   Usar los datos capturados en la lógica de la aplicación.

✅ Unidad 5 — Componentes de selección

🔘 RadioButton — Selección única

Permite elegir una sola opción entre varias, agrupadas mediante un **RadioGroup**.

Propiedades importantes

- Solo puede haber **una opción activa por grupo**.
- Seleccionar una desactiva automáticamente las demás.
- Ideal para preferencias o configuraciones exclusivas.

```
<RadioGroup
    android:layout_width="match_parent"
    android:layout_height="wrap_content"
    android:orientation="vertical">

    <RadioButton
        android:id="@+id/rbOpcion1"
```

```
        android:text="Primera opción"/>

    <RadioButton
        android:id="@+id/rbOpcion2"
        android:text="Segunda opción"/>
</RadioGroup>
```

☑️ CheckBox — Selección múltiple

- Cada *CheckBox* funciona de forma **independiente**.
- Permite **varias selecciones simultáneas**.
- Perfecto para activar o desactivar funcionalidades.

⚙️ Eventos y validación de selecciones

- `OnCheckedChangeListener` → detecta cambios en *RadioButton* o *CheckBox*.
- `isChecked()` → comprueba el estado actual del componente.

```
if (checkBox.isChecked()) {
    procesarOpcion();
}
```

🔀 Unidad 6 — Switch: Control de estados

🌓 Descripción general

El componente **Switch** representa un **estado binario (ON/OFF)** visual y funcional, ideal para activar o desactivar características.

Estados

- **ON:** funcionalidad activa, color destacado, deslizador a la derecha.
- **OFF:** funcionalidad inactiva, color neutro, deslizador a la izquierda.

🛠️ Eventos y personalización

1. `setOnCheckedChangeListener`
   Ejecuta acciones al cambiar de estado.
2. **Personalización visual**
   Permite modificar colores, textos ("On"/"Off"), tamaños y estilos.
3. **Control por código**
   - `setChecked(true/false)` → cambia el estado.
   - `isChecked()` → consulta el estado actual.

🖼️ Unidad 7 — Trabajando con imágenes

🧩 ImageView — Mostrar imágenes

- Componente versátil para mostrar imágenes estáticas desde:

    - Recursos locales (`drawable`)
    - Archivos (`assets`)
    - URLs remotas
- Admite múltiples formatos: PNG, JPG, GIF, WebP.
- Escalado automático con `scaleType` .

```
<ImageView
    android:layout_width="200dp"
    android:layout_height="200dp"
    android:src="@drawable/imagen"
    android:scaleType="centerCrop" />
```

🔴 ImageButton — Botones con imagen

- Combina las funciones de *Button* e *ImageView*.
- Cambia su apariencia según el estado (*pressed*, *focused*, *normal*).
- Soporta el evento `onClick` igual que un botón convencional.

📁 Carpetas *drawable* y resolución

Android selecciona automáticamente la versión más adecuada según la **densidad de pantalla (dpi):**

1. `drawable-xxxhdpi`
2. `drawable-xxhdpi`
3. `drawable-xhdpi`
4. `drawable-hdpi`
5. `drawable-mdpi`

Esto optimiza tanto la **calidad visual** como el **uso de memoria**.

📋 Unidad 8 — Listas y selección

🔽 Spinner — Lista desplegable

- Control compacto que muestra una lista al tocarlo.
- Ideal para **selección única** con espacio reducido.

Ventajas

- Ahorra espacio en pantalla.
- Fácil de usar y configurar con un **ArrayAdapter**.

📜 ListView — Listas desplazables

- Utiliza el patrón **Adapter**, que conecta los datos con la presentación visual.
- **Reutiliza vistas** fuera de pantalla (optimización de memoria).
- Permite gestionar eventos de clic con `OnItemClickListener` .

⚙️ ArrayAdapter — Conexión entre datos y vista

Ejemplo básico:

```
String[] opciones = {"Opción 1", "Opción 2", "Opción 3"};

ArrayAdapter<String> adapter = new ArrayAdapter<>(
    this,
    android.R.layout.simple_spinner_item,
    opciones
);

adapter.setDropDownViewResource(android.R.layout.simple_spinner_dropdown_item);
spinner.setAdapter(adapter);
```

> 📌 El *ArrayAdapter* actúa como **puente entre los datos y el componente visual**, manejando automáticamente la presentación y la selección.

⚙️ Unidad 9 — Profundizando en *Activities*

🔁 Ciclo de vida completo de una Activity

Método	Descripción
`onCreate()`	Inicialización y configuración de layout.
`onStart()`	La Activity se vuelve visible.
`onResume()`	La Activity pasa a primer plano (interacción activa).
`onPause()`	Pierde el foco (se pausa).
`onStop()`	Ya no es visible, libera recursos no esenciales.
`onDestroy()`	Finaliza completamente y limpia la memoria.

📤 Intents — Navegación avanzada

Tipo	Uso	Ejemplo
Explícito	Navegación interna	Abrir otra pantalla de la app
Implícito	Acción del sistema	Enviar email, abrir web
Con datos	Transferencia de información	`putExtra()`
Para resultado	Espera respuesta	`startActivityForResult()`

📨 Intents: Mensajería entre componentes

En Android, los **Intents** son objetos que permiten **comunicar diferentes componentes** de una aplicación (o incluso entre aplicaciones distintas).
Sirven para **iniciar Activities**, **enviar datos**, **abrir páginas web**, **hacer llamadas**, etc.

🔹 Intents Explícitos

- **Definen exactamente qué componente se va a ejecutar.**
- Se usan para **navegar entre Activities dentro de la misma aplicación.**
- Ejemplo: abrir una segunda pantalla llamada `SecondActivity`.

```
Intent intent = new Intent(this, SecondActivity.class);
startActivity(intent);
```

➡️ El programador indica directamente el destino (`SecondActivity.class`).

🔹 Intents Implícitos

- **No especifican el componente exacto**, sino **la acción que se desea realizar.**
- El **sistema operativo decide** qué aplicación puede manejar esa acción.
- Ejemplo: abrir una página web en el navegador predeterminado.

```
Intent intent = new Intent(Intent.ACTION_VIEW, Uri.parse("https://google.com"));
startActivity(intent);
```

➡️ Android detecta que la acción es “ver una URL” y lanza el navegador automáticamente.

🧠 **Resumen rápido:**

- **Explícito** → comunicación interna (dentro de la app).
- **Implícito** → comunicación externa (otras apps o servicios del sistema).

🧭 Unidad 10 — Layouts: Organización de la interfaz

📐 Tipos de layouts fundamentales

Tipo	Descripción	Uso típico
LinearLayout	Organiza elementos en línea horizontal o vertical.	Formularios simples.
TableLayout	Distribuye elementos en filas y columnas.	Formularios o tablas.
FrameLayout	Superpone elementos.	Contenedor de *fragments*.
ConstraintLayout	Sistema flexible y moderno de restricciones.	Interfaces complejas y responsivas.

🧩 ConstraintLayout — Diseño moderno

Ventajas

- Mayor rendimiento.
- Menos anidación de layouts.
- Editor visual potente.
- Diseño responsive automático.
- Permite *chains* y *barriers* para organización avanzada.

Restricciones básicas

Cada elemento necesita al menos **una restricción horizontal y una vertical**:

```
app:layout_constraintTop_toTopOf="parent"
app:layout_constraintStart_toEndOf="@id/otroElemento"
app:layout_constraintBottom_toBottomOf="parent"
app:layout_constraintEnd_toStartOf="@id/otroElemento"
```

🚀 Cierre del módulo — Conclusión

> “Tu viaje en Android continúa.”

- **Siguientes pasos:**
  - Aprender fragmentos, bases de datos y conexión a red.
  - Aplicar arquitectura **MVVM**.
  - Practicar con proyectos personales.
- **Consejos del profesor:**
  - Programar constantemente.
  - Contribuir en proyectos open source.
  - Participar en comunidades (Stack Overflow, GitHub…).
